UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/687,233	10/15/2003	Martin Runte	6570P015	8107

45062 7590 01/28/2008
SAP/BLAKELY
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
01/28/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

mn

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

MAILED

JAN 25 2008

Technology Center 2100

Application Number: 10/687,233
Filing Date: October 15, 2003
Appellant(s): RUNTE ET AL.

Vincent H. Anderson (Reg. No. 54,962)
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed on November 5, 2007 appealing from the Office action mailed on June 4, 2007.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The Examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The Appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The Appellant's statement of the grounds of rejection to be reviewed on appeal is substantially correct. The changes are as follows:

- Claims 36 and37 were cancelled and were not rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,519,767 of Carter et al.
- Claim 38 was cancelled and was not rejected under 35 U.S.C. § 103(a) as being unpatentable over Carter et al. in view of U.S. Patent No. 6,658,659 of Hiller et al.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

6,519,767	CARTER et al.	2-2003
6,658,659	HILLER et al.	12-2003

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

1. **Claims 1-4, 6-20, 22-25, and 32-34** are rejected under 35 U.S.C. 102(e) as being anticipated by **<u>Carter et al.</u> (US 6,519,767)**.

As per **Claim 1**, <u>Carter et al.</u> disclose:

- automatically detecting a change introduced into a software object of a first software subsystem, wherein the software object is used by software objects of a second software subsystem *(see Column 8: 12-17, "Version compatibility analyzer 70 utilizes a process 76* *illustrated in FIG. 6 and described below to detect any modifications ("version incompatible differences") from existing object server 66 that prevent compiler 52 from building new object server 64 so as to be version compatible with existing object server 66."; Column 14: 9-16, "... version compatibility analyzer 70 further compares the types of the classes and public members (including the member's parameters and return values). If the types of any of these identically named classes or public members do not match, new object server 64 is considered to have changed the class or public member over existing object server 66.")*;

- determining whether the change is compatible with the software objects of the second software subsystem *(see Column 14: 17-23, "... when new object server 64 is determined to have removed or changed types of any class or public member over existing object server 66 from the comparison step in 161, new object server 64 is determined by version compatibility analyzer 70 to be incompatible with existing object server 66.")*; and

- implementing the introduced change to generate an updated software object if the change is compatible with the software objects of the second software subsystem without introducing any changes into the software objects of the second software subsystem *(see Column 8: 21-25, "When no version incompatible differences are detected by version compatibility analyzer 70, compatible object server generator 72 generates the interface related information in process 78 so as to be version compatible with existing object server 66."; Column 14: 33-38, "when new object server 64 is determined from the step 161 comparison to have added any class or any public member to a class over existing object server 66, new object server 64 is determined by version compatibility analyzer 70 to be version compatible with existing object* *server 66. In this situation, new object server 64 can be built to support each interface of existing object server 66 ... ")*;

- otherwise, rejecting the introduced change and generating an error notification *(see Column 14: 25-28, "... user interface 59 notifies the user that new object server 64 cannot be compatible, such as by displaying a dialog box with such a message.")*.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Carter et al. further disclose:

- wherein determining whether the change is compatible further comprises determining whether the change is predefined as compatible *(see Column 13: 10-18, "For example, the following edits do not prevent new object server 64 from being version compatible with existing object server 66: changing code within a member, changing the position of a member in a class, and adding a new member to a class.")*.

As per **Claim 3**, the rejection of **Claim 2** is incorporated; and Carter et al. further disclose:

- allowing the change if the change is predefined as compatible *(see Column 13: 10-13, "... edits which do not change any interface supported by existing object server 66 can be made to source data 60 without making new object server 64 incompatible.")*.

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Carter et al. further disclose:

- issuing a message that the change is not allowed if the change is not predefined as compatible *(see Column 14: 25-28, "... user interface 59 notifies the user that new object server 64 cannot be compatible, such as by displaying a dialog box with such a message.")*.

As per **Claim 6**, Carter et al. disclose:

- identifying a subset of software objects of a first software subsystem and declaring the subset of software objects frozen *(see Column 8: 30-37, "In the existing version object server 92, clock object 94 has three members 96-98 (FIG. 3A), respectively named "SetTime," "GetHour," and "GetMinute," as well as properties 100 which include integer values named "hour," and "minute." In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.")*; and

- detecting a change introduced into a frozen software object from the subset of software objects; and prior to allowing the change *(see Column 8: 12-17, "Version compatibility analyzer 70 utilizes a process 76 illustrated in FIG. 6 and described below to detect any modifications ("version incompatible differences") from existing object server 66 that prevent compiler 52 from building new object server 64 so as to be version compatible with existing object server 66."; Column 14: 9-16, "... version compatibility analyzer 70 further compares the types of the classes and public members (including the member's parameters and return values). If the types of any of these identically named classes or public members do not match, new object* *server 64 is considered to have changed the class or public member over existing object server 66.")*,

- determining with a compatibility check whether the change is compatible with a second software subsystem *(see Column 14: 17-23, "... when new object server 64 is determined to have removed or changed types of any class or public member over existing object server 66 from the comparison step in 161, new object server 64 is determined by version compatibility analyzer 70 to be incompatible with existing object server 66.")*; and

- issuing a notice indicating results of the compatibility check *(see Column 14: 25-28, "... user interface 59 notifies the user that new object server 64 cannot be compatible, such as by displaying a dialog box with such a message.")*.

As per **Claim 7**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein the subset of software objects declared frozen includes software objects of the first software subsystem that are used by the second software subsystem *(see Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.")*.

As per **Claim 8**, the rejection of **Claim 7** is incorporated; and Carter et al. further disclose:

- wherein frozen software objects are classified to include released objects and restricted objects *(see Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.").*

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and <u>Carter et al.</u> further disclose:

- wherein the released objects include software objects that are used by the second software subsystem without restrictions *(see Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.").*

As per **Claim 10**, the rejection of **Claim 8** is incorporated; and <u>Carter et al.</u> further disclose:

- wherein the restricted objects include software objects that are used by software objects of the second software subsystem, the software objects being fewer in number than a threshold *(see Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values,* *"alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.").*

As per **Claim 11**, the rejection of **Claim 8** is incorporated; and Carter et al. further disclose:

- wherein an identification of recent changes introduced into a restricted object is provided when software objects of the second software subsystem request new usage of the restricted object *(see Column 8: 55-59, "In this vtable, ITime$_1$ interface 110 includes pointers to members 96-98 of clock object 94. The members 96-98 can then be accessed by code in client application 101 based on their particular offset within the vtable.").*

As per **Claim 12**, the rejection of **Claim 8** is incorporated; and Carter et al. further disclose:

- wherein classification of the frozen software objects is based on a number of times a frozen software object is used by the second software subsystem *(see Column 9: 1-6, "AddRef and Release functions 125-126 include code for OLE 2 to track the number of references from client applications currently using an instance of clock object so that memory allocated to such instance can be released when the instance is no longer in use.").*

As per **Claim 13**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein a software object is a function module *(see Column 7: 13-16, "Integrated development environment 50 further includes a source editor 56 which provides conventional editing tools for a user to enter and modify programs, including programs which are to be compiled into object servers.").*

As per **Claim 14**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein a software object is a data structure *(see Column 7: 27-31, "Source data 60 of the program can be stored by source editor 56 in the form of a data stream representation of the programming language statements, or more preferably a data stream consisting of a tokenized representation of the programming language statements.").*

As per **Claim 15**, the rejection of **Claim 13** is incorporated; and Carter et al. further disclose:

- wherein the software object includes an environment of the function module *(see Column 7: 13-16, "Integrated development environment 50 further includes a source editor 56 which provides conventional editing tools for a user to enter and modify programs, including programs which are to be compiled into object servers.").*

As per **Claim 16**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein a software object includes a class and an environment of the class *(see Column 7: 25-27, "In the Visual Basic language system, each project can include one or more class modules which each define an object.").*

As per **Claim 17**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein a software object includes an interface and an environment of the interface *(see Column 7: 25-27, "In the Visual Basic language system, each project can include one or more class modules which each define an object.").*

As per **Claim 18**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein a software object includes a program and an environment of the program *(see Column 7: 23-25, "... each program is entered by the user in the form of a "project" in the Visual Basic language system.").*

As per **Claim 19**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein the detecting the change comprises automatically monitoring development of software code *(see Figure 6; Column 13: 36-46, "In process 76, version compatibility analyzer 70 determines version compatibility with existing object server 66 by comparing source data 60 to type information stored in existing object server's type library 150.").*

As per **Claim 20**, the rejection of **Claim 6** is incorporated; and Carter et al. further disclose:

- wherein the determining whether the change is compatible comprises determining whether there is a predefined declaration of compatibility of the change *(see Column 13: 10-18, "For example, the following edits do not prevent new object server 64 from being version compatible with existing object server 66: changing code within a member, changing the position of a member in a class, and adding a new member to a class.")*.

As per **Claim 22**, Carter et al. disclose:

- performing a global compatibility check of software objects of a first software subsystem by determining whether any changes were introduced into a subset of the software objects of the first software subsystem since the time of a last compatibility check, wherein the introduced changes were introduced without obtaining prior approval *(see Figure 6; Column 12: 62-67, "FIG. 6 shows version compatibility analysis process 76 (FIG. 6) comprising steps 160-167 which is utilized by compiler 52 (FIG. 2) in version compatibility analyzer 70 (FIG. 2) to determine whether new object server 64 (FIG. 2) can be version compatible with existing object server 66 (FIG. 2).")*;

- identifying software objects of a second software subsystem affected by an unapproved change, wherein the affected software objects of the second software system are software objects using at least one software object of the subset of the software objects of the first software system *(see Figure 6: 164 and 165; Column 7: 60-65, "The user also can select* *with user input 58 to have compiler 52 compile source data 60 utilizing the process according to the present invention for automatically building a version compatible object server, or in the conventional manner (without the version compatible object server building process).";* *Column 8: 3-8, "If no file name is specified in the Compatible Object Application field (such as when compiling the first version of existing object server 66), compiler 52 compiles source data 60 conventionally into an object server without the automatic version compatible object server building process."; Column 14: 33-38, "As indicated at steps 164 and 165, when new object server 64 is determined from the step 161 comparison to have added any class or any public member to a class over existing object server 66, new object server 64 is determined by version compatibility analyzer 70 to be version compatible with existing object server 66.")*; and

- issuing a notice of possible incompatibility between affected software objects and software objects including the unapproved change *(see Column 14: 25-28, "... user interface 59 notifies the user that new object server 64 cannot be compatible, such as by displaying a dialog box with such a message.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and Carter et al. further disclose:

- wherein the performing a global compatibility check comprises comparing a current version of software code with a version of the software code at the time of a last global compatibility check *(see Column 12: 62-67, "FIG. 6 shows version compatibility analysis process 76 (FIG. 6) comprising steps 160-167 which is utilized by compiler 52 (FIG. 2) in* *version compatibility analyzer 70 (FIG. 2) to determine whether new object server 64 (FIG. 2) can be version compatible with existing object server 66 (FIG. 2).").*

As per **Claim 24**, the rejection of **Claim 22** is incorporated; and Carter et al. further disclose:

- wherein the subset of the software objects includes frozen software objects *(see Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.").*

As per **Claim 25**, the rejection of **Claim 24** is incorporated; and Carter et al. further disclose:

- wherein the frozen software objects include software objects of the first software subsystem used by software objects of the second software subsystem *(see Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.").*

Claims 32-34 are article of manufacture claims corresponding to the method claims above (Claims 1, 2, and 4) and, therefore, are rejected for the same reasons set forth in the rejections of Claims 1, 2, and 4.

2. **Claims 5, 21, and 35** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Carter et al. (US 6,519,767)** in view of **Hiller et al. (US 6,658,659)**.

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Carter et al. do not disclose:

- allowing the change if an expert declares the change compatible upon receiving a request for a manual compatibility check, wherein the change is not predefined as compatible.

Hiller et al. disclose:

- allowing the change if an expert declares the change compatible upon receiving a request for a manual compatibility check, wherein the change is not predefined as compatible *(see Column 11: 21-25, "... allow a programmer to designate acceptable compatibility according to a minimum version number or a version number corresponding to a minimum date of release.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Hiller et al. into the teaching of Carter et al. to include allowing the change if an expert declares the change compatible upon receiving a request for a manual compatibility check, wherein the change is not predefined as compatible. The modification would be obvious because one of ordinary skill in the art would be motivated to select a suitable version of a program to avoid version incompatibility problems as new versions are released *(see Hiller et al. – Column 13: 6-12).*

As per **Claim 21**, the rejection of **Claim 7** is incorporated; however, Carter et al. do not disclose:

- wherein the determining whether the change is compatible comprises determining whether an expert declared the change compatible.

Hiller et al. disclose:

- wherein the determining whether the change is compatible comprises determining whether an expert declared the change compatible *(see Column 11: 21-25, "... allow a programmer to designate acceptable compatibility according to a minimum version number or a version number corresponding to a minimum date of release.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Hiller et al. into the teaching of Carter et al. to include wherein the determining whether the change is compatible comprises determining whether an expert declared the change compatible. The modification would be obvious because one of ordinary skill in the art would be motivated to select a suitable version of a program to avoid version incompatibility problems as new versions are released *(see Hiller et al. – Column 13: 6-12).*

Claim 35 is rejected for the same reason set forth in the rejection of Claim 5.

(10) Response to Argument

A. The rejections of Claims 1-4 and 32-34 under 35 U.S.C. § 102(e) for being anticipated by Carter et al. (hereinafter "Carter")

In the Appeal Brief, Appellant argues:

a) Appellants note the significant absence of any detecting of a change to a software object, in contrast to what is recited in Appellants' claims. As appears from the cited reference, Carter fails to consider changes at all to a software object, and is rather concerned with changes to interfaces to the applications (specifically, the object servers).

(See Appeal Brief – page 6 to page 7)

Examiner's response:

a) Examiner disagrees. Carter clearly discloses detecting changes to a software object *(see Figure 2: 150 and 170; Figure 6: 162 and 164; Column 8: 12-17, "Version compatibility analyzer 70 utilizes a process 76 illustrated in FIG. 6 and described below to detect any modifications ("version incompatible differences") from existing object server 66 that prevent compiler 52 from building new object server 64 so as to be version compatible with existing object server 66."; Column 14: 9-16, "... version compatibility analyzer 70 further compares the types of the classes and public members (including the member's parameters and return values). If the types of any of these identically named classes or public members do not match, new object server 64 is considered to have changed the class or public member over existing object server 66.")*. Thus, the version compatibility analyzer compares the type information of each class (software object) in the new object server to the type information of an identically named class in the existing object server's type library to identify any class changes.

As the claims are interpreted as broadly as their terms reasonably allow (see MPEP § 2111.01 I) and also in light of the non-limiting exemplary definition for the term "object" in paragraph [0016] of the specification to include "function modules, programs, data objects, classes, class components, interfaces, attributes, etc.," the interpretation of a broad limitation of a "software object" as a program or a class or an interface and the like by one of ordinary skill in the art is considered to be reasonable and logical.

In the Appeal Brief, Appellant argues:

b) As a first matter, such a broad interpretation of the expression "software objects" is contrary to the way that term is used by the reference itself. Carter at col. 1, lines 17 to 25 and 33 to 43 recites the following:

> The term "object" generally refers to an instance of a programmer-defined data structure (the data or variables of which are referred to as the object's "properties") and functions that manipulate that structure (referred to as "member functions," "member procedures," or simply "members" of the object). In other words, objects are a combination of a data structure with a set of functions that perform methods on the data structure. The term "class" generally refers to the definition of an object ....
>
> One benefit to the use of objects is that their members define a standard way for other programs to interact with or access the object's properties. A set of semantically related functions implemented on an object is generally referred to as an "interface" of the object. Each object typically includes at least one interface, but in some object-oriented systems (such as OLE 2 described below) may include more than one interface. By allowing access to an object's members through an interface, the object effectively "exposes" its functionality for use by other application programs ....

As shown in the reference itself, the term object is defined in such a way that a standalone application is excluded from its definition. In fact, objects and "application programs" are discussed as being on conceptually different levels, as one of skill in the art would understand them to be. Appellants submit that interpreting "software objects" to include application programs would be inconsistent with the definitions provided in Carter itself.

(See Appeal Brief – page 7 to page 8)

Examiner's response:

b) Carter at Column 1, lines 13-17 discloses the following:

> A significant development in the computer programming field is that of object oriented programming. Object-oriented programming generally refers to computer programs, programming languages and tools which utilize the concept of "objects."

Thus, Carter defines an "object" as a feature of the object-oriented programming paradigm, which is what the term is typically defined as in the computing art. Examiner agrees with Appellant's argument that a standalone application is excluded from the definition of an "object." However, Examiner's interpretation of the claim limitation of a "software object" as an "object server" is consistent with the definition of "object server" provided by Carter in Column 1: 50-65:

> Microsoft Corporation's Object Linking and Embedding 2 ("OLE 2") provides a system for creating object-oriented applications for use with its Windows operating system. <u>OLE 2 provides a way for objects supplied by one application program (hereafter referred to as the "object application" or "object server")</u> (emphasis added) to be controlled by another application program (hereafter referred to as a "controlling application" or "client application"). <u>After an object server is created, its programmer or developer can distribute the object server in the form of an executable file for use by others. Other developers and</u>

> <u>end users can then create client applications which make use of the functionality embodied in the object server's objects by accessing those objects through their interfaces</u> (emphasis added). Accordingly, OLE 2 provides a way for the functionality embodied in the object server to be useable by any number of client applications.

As shown, Carter discloses that an application program is referred to as "object application" or "object server." This clearly indicates that an "object server" is an application program containing objects and is <u>not</u> an object of a class as averred by the Appellant.

In the Appeal Brief, Appellant argues:

c) Furthermore, assuming the application programs of Carter are somehow interpreted as software objects, the Office has failed to point to what is purported to be the "subsystems" as claimed. A claim rejection is inadequate when it fails to show the same detail as what is claimed. See MPEP 706. The object servers are different versions of each other running on the same software environment. There is no assertion of different software subsystems. If the object server is the object, it cannot be the subsystem, because then it would be an object within itself as both the subsystem and object. Thus, there is a logical inconsistency with the interpretation in the Office Actions.

(See Appeal Brief – page 8)

Examiner's response:

c) Examiner disagrees. Carter clearly discloses subsystems *(see Figure 2: 64 and 66)*. Thus, Elements 64 and 66 of Figure 2 illustrate a new object server executable file and an existing object server executable file, respectively. Examiner agrees with Appellant's argument that the object servers are different versions of each other running on the same software environment. However, paragraph [0016] of the specification clearly indicates that the subsystems are independent subsystems (different versions of the object server executable file), which may or may not be located on several machines.

Notwithstanding Appellant's argument with respect to the object server cannot be the subsystem if it is the object, the claim language does not define any specific relationship between the software object and the software subsystem. Nor does the claim language define the software object and the software subsystem as separate entities. The claims merely recite a software object of a software subsystem without any further clarification on how the software object relates to the software subsystem.

In the Appeal Brief, Appellant argues:

d) Appellants note that one of skill in the art would appreciate that there are two separate meanings for the term "program." One definition is consistent with Carter, which is that an executable file (an application) is sometimes referred to as a program. However, a program does not necessarily mean an application program, and thus the term "program" can have at least one meaning not consistent with Carter.

For example, a second, distinct definition of a program is referred to in Appellants' paragraph [0016], referring to a function module, routine, or subroutine callable in an application (not being the application itself), to perform a certain function, algorithm, method, etc. The second definition is consistent with Appellants' Specification, which refers to programs in the same context as "classes, class components, [and] attributes." The first definition would be inconsistent with a discussion of "classes, class components, [and] attributes." The term program in Appellants' Specification appears only in paragraph [0016], and the claims. There is no evidence to assume that program means an application program as referenced in Appellants' Specification.

(See Appeal Brief – page 8 to page 9)

Examiner's response:

d) Examiner disagrees. It is noted that paragraph [0016] of the specification only provides an exemplary definition for the term "object," not the term "program" as averred by the Appellant. Thus, Appellant's purported second definition for the term "program" is unsupported. In attempting to challenge the merit of Carter, Appellant has failed to present arguments as to why one of ordinary skill in the art would not associate the term "program" with the term "application program."

In the Appeal Brief, Appellant argues:

e) Appellants submit that the reasoning in the Office Action equating "program" to the claimed "objects" is similar to the following hypothetical situation. Consider a patent application that made reference to "pointing devices such as a mouse, a trackball, or a touchpad." Then consider that a hypothetical Office Action cited a reference that referred to a rodent, and specifically a mouse. The hypothetical Office Action may proffer that one of skill in the art would understand that a mouse can refer to a pointing device, which means the rodent discloses the pointing device. Appellants emphasize that the mere fact that the terms ("mouse") are identical on their face does not necessarily mean they have the same meaning. While this example may be slightly exaggerated, the Office Action in the present case has similarly, improperly equated a term (program, meaning executable file) with another term (software object, which may include a type of program or function module) that is not a direct equivalent. Thus, Appellants respectfully submit that the Carter reference fails to support the rejection in the Office Actions. The reasoning of the Final Office Action is not logical, and the application of the Carter reference to the claims is misplaced.

(See Appeal Brief – page 9)

Examiner's response:

e) Examiner disagrees. A pointing device called "mouse" and a rodent called "mouse" are completely different subject matters and out of context with each other. This is <u>not</u> similar to the situation at present where the subject matters belong to the same art—namely, the computing art. The invention of Carter is in the same field of endeavor as the Appellant because both are concerned with building version compatible software. The computer terms used in Carter and by the Appellant might not be directly equivalent, but various interpretations of these computer terms are justifiable as long as they are sound and reasonable within the context of computing art. The reasoning applied by the Examiner in interpreting these terms is within the context of computing art and hence, reasonable to one of ordinary skill in the computing art.

In the Appeal Brief, Appellant argues:

f) As Appellants stated in a previous Response, "Without conceding that Carter's application servers are the same as the objects recited in the independent claims, Appellants submit that the rejection is defective. The Final Office Action fails to make any assertion that objects of different subsystems are disclosed in the cited reference." The Advisory Action fails to address this defect of the rejection, and fails to address Appellants' argument. Thus, the Advisory Action is incomplete and non-responsive, and the arguments made by Appellant have not all been addressed by the Office, denying Appellants the opportunity to be heard.

(See Appeal Brief – page 9 to page 10)

Examiner's response:

f) Examiner disagrees. Examiner clearly noted all of Appellant's arguments on the Advisory Action (mailed on 08/14/2007). Appellant did not present any specific arguments regarding "software subsystems" or how the applied prior art fails to disclose "software subsystems." Appellant's arguments in the Amendment After Final (filed on 08/01/2007) are directed to disagreeing with Examiner's line of reasoning for interpreting an "executable file" as a "software object." *(See Remarks – page 8 to page 9).* Examiner responded to the arguments in the "Examiner's response" section of the Advisory Action. Moreover, Appellant's assertion of the Final Rejection as being failing to make any assertion that objects of different subsystems are disclosed in the cited reference is a mere allegation because Appellant has failed to specifically point out how the language of the claims patentably distinguishes them from the references and therefore, failed to comply with 37 CFR 1.111(b). Examiner has responded to every argument presented by the Appellant in the Advisory Action. Thus, the Advisory Action is complete and responsive.

<u>B. The rejections of Claims 6, 7, and 13-20 under 35 U.S.C. § 102(e) for being anticipated by Carter</u>

In the Appeal Brief, Appellant argues:

a) The Office Actions attempt to reject such features under col. 13, line 47 to col. 14, line 32 of Carter. That section of Carter describes comparing identically named classes of the updated object server with the previous version of the object server. Thus, there are no "frozen objects" as claimed, which are objects identified as software objects of a first software subsystem used in a second subsystem. See also, paragraph [0016] of Appellants' Specification.

(See Appeal Brief – page 10)

Examiner's response:

a) Examiner disagrees. Examiner relied upon Column 8, lines 30-37 of Carter to reject the feature of "frozen objects," <u>not</u> Column 13, line 47 to Column 14, line 32 as pointed out by the Appellant. Carter clearly discloses the feature of "frozen objects" *(see Column 8: 30-37, "In the existing version object server 92, clock object 94 has three members 96-98 (FIG. 3A), respectively named "SetTime," "GetHour," and "GetMinute," as well as properties 100 which include integer values named "hour," and "minute." In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new* *integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.").* Thus, the "SetTime," "GetHour," and "GetMinute" member functions of the existing object server's clock object are used in the new object server's clock object (objects identified as software objects of a first software subsystem used in a second software subsystem).

C. The rejections of Claims 8, 11, and 12 under 35 U.S.C. § 102(e) for being anticipated by Carter

In the Appeal Brief, Appellant argues:

a) The discussion above with respect to the rejection of claims 6 and 7 is similar to the line of reasoning proffered by the Office with respect to these claims. That is, the Office recites different versions of elements of the object server as showing the released and restricted objects. The argument in the Office Action is not consistent with its declaration that the object servers are the software objects. Furthermore, even assuming the reference discloses software objects, the Office has failed to assert frozen software objects classified as released and restricted objects.

(See Appeal Brief – page 11)

Examiner's response:

a) Examiner disagrees. Carter clearly discloses released objects and restricted objects. See Examiner's responses in Sections D and E below.

<u>D. The rejection of Claim 9 under 35 U.S.C. § 102(e) for being anticipated by Carter</u>

In the Appeal Brief, Appellant argues:

a) Claim 9 further recites limitations directed to released objects including software objects used by the second software subsystem without restrictions. The Office merely points to the same section of the Carter reference used to reject claims 6, 7, and 8. The Office makes no attempts to provide reasoning to explain how the classes of the object servers are supposedly used with or without restrictions. The Office fails to establish that such a feature is either expressly or inherently present in the cited reference.

(See Appeal Brief – page 11)

Examiner's response:

a) Examiner disagrees. Carter clearly discloses software objects that are used by the second software subsystem without restrictions *(see Figure 3B: 96-98 and 124-129; Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.")*. Thus, the new object server's clock object contains the same member functions of the existing object server's clock object, such as "SetTime," "GetHour," and "GetMinute." Therefore, there is no restriction placed upon these member functions to be used by the new object server.

E. The rejection of Claim 10 under 35 U.S.C. § 102(e) for being anticipated by Carter

In the Appeal Brief, Appellant argues:

a) Claim 9 (sic) further recites limitations directed to restricted objects including software objects used by the second software subsystem fewer than a threshold number of times, or a number of the objects being lower than a threshold (i.e., a number of instances). The Office merely points to the same section of the Carter reference used to reject claims 6, 7, and 8. The Office makes no attempts to provide reasoning to explain how the classes of the object servers are supposedly restricted in numbers of instances or not. The Office fails to establish that such a feature is either expressly or inherently present in the cited reference.

(See Appeal Brief – page 12)

Examiner's response:

a) Examiner disagrees. Carter clearly discloses software objects that are used by software objects of the second software subsystem, the software objects being fewer in number than a threshold *(see Figure 3B: 100; Column 8: 30-37, "In the revised version object server 92', clock object 94' is changed by adding a new member 99, named "SetAlarm," along with new integer values, "alarmHour," and "alarmMinute" to properties 100'. A client application 101 (FIG. 3) manipulates or controls clock object 94 or 94' by calling members 96-99.")*. Thus, the properties of the clock object of the new object server are used by the clock object's member functions. The number of member variables (*e.g.*, "alarmHour" and "alarmMinute") added to the properties must depend on the size of the data structure (fewer in number than a threshold).

Furthermore, the claim language does not limit the scope of "threshold" to number of times or number of instances. Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993). Thus, as the claims are interpreted as broadly as their terms reasonably allow (see MPEP § 2111.01 I), the interpretation of a broad limitation of a "threshold" as the size of a data structure and the like by one of ordinary skill in the art is considered to be reasonable by its plain meaning.

F. The rejections of Claims 22-25 under 35 U.S.C. § 102(e) for being anticipated by Carter

In the Appeal Brief, Appellant argues:

a) As a first matter, Appellants submit that the reasoning provided above in subsection A with respect to claim 1 applies equally well to claim 22 and its dependents. Furthermore, regarding the other feature pointed out, Appellants note that claim 22 implies that changes may be made without a compatibility check being performed, or rather, that the timing of a compatibility check and the timing of a change to a software object do not necessarily coincide. Such a feature is absent in the cited reference, which requires a compatibility check each time the application program is changed.

(See Appeal Brief – page 12 to page 13)

Examiner's response:

a) Examiner disagrees. Carter clearly discloses performing a global compatibility check of software objects of a first software subsystem by determining whether any changes were introduced into a subset of the software objects of the first software subsystem since the time of a last compatibility check, wherein the introduced changes were introduced without obtaining prior approval *(see Figure 6; Column 12: 62-67, "FIG. 6 shows version compatibility analysis process 76 (FIG. 6) comprising steps 160-167 which is utilized by compiler 52 (FIG. 2) in version compatibility analyzer 70 (FIG. 2) to determine whether new object server 64 (FIG. 2) can be version compatible with existing object server 66 (FIG. 2).")*. Thus, prior to commencing the version compatibility analysis process, there are changes that already exist between the existing object server and the new object server. These changes were introduced without obtaining prior approval. Otherwise, the version compatibility analysis would not be required because the changes made to the new object server would be compatible with the existing object server.

Furthermore, with respect to Appellant's arguments that the claim implies that changes may be made without a compatibility check being performed or that the timing of a compatibility check and the timing of a change to a software object do not necessarily coincide, it is noted that these limitations are only implied and not explicitly recited in the claim and therefore, warrant no patentable weight. Thus, broadest reasonable interpretations of the broadly claimed limitations are deemed to be proper.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the Examiner in the Related Appeals and Interferences section of this Examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Qing Chen

Conferees:

Wei Zhen

Eddie Lee

WEI ZHEN
SUPERVISORY PATENT EXAMINER

[illegible]

[illegible]

EDDIE C. LEE
SUPERVISORY PATENT EXAMINER